CLMSR: A Tick-Based Implementation of the Continuous Logarithmic Market Scoring Rule for Range Prediction Markets

Signals Research Team
jaegun@signals.wtf, danny@signals.wtf
Signals Protocol

Version 1.0 — September 21, 2025

Abstract

We specify **CLMSR**, a practical on-chain realization of the Continuous Logarithmic Market Scoring Rule over a ticked (discretized) outcome line. Traders buy and sell *range securities* that pay if the realized outcome falls within a chosen tick interval. The mechanism preserves standard LMSR guarantees—path-independent cost and the identity *price = probability*—while remaining efficient on-chain via a lazy multiplicative segment tree with $O(\log n)$ range updates and queries.

Our contributions are threefold. First, we state a clean tick semantics (half-open $[L, U)$) and show that range trades reduce to a two-partition LMSR with masses $(W_B, \mathcal{Z} - W_B)$, clarifying both intuition and gas-minimal implementation. Second, we enforce a per-trade single conversion at the U6/WAD (18) boundary; the rounding direction is an implementation parameter (we recommend a maker-tilted variant and provide bounds). Third, we design a lazy segment tree whose invariants keep the cached partition sum $\mathcal{Z}$ *consistently exact under WAD* under lazy propagation, yielding numerically robust costs and prices.

We argue that in continuous-outcome markets implemented as disjoint tick order books with range synthesis via routing, enforcing normalization and additivity requires a central potential; CLMSR provides this on-chain while retaining bounded loss and on-chain viability. Externally we account in a **USDC-style 6-decimal ERC-20** (U6); internally, all math uses **WAD (18)**. Operational matters (governance, tokenomics, off-chain infra) are out of scope.

1 Introduction

Prediction markets aggregate dispersed beliefs about uncertain future events by letting participants buy and sell contingent claims. Prices then encode information: in a well-designed mechanism, the instantaneous price of a claim can be read as the market-implied probability of its payoff. On chain, the mechanism must remain path-independent, capital-efficient for the maker, and computationally light.

A practical way to satisfy these constraints is to replace an order book with a *market-scoring rule* (MSR). An MSR is specified by a convex *cost function* $C(\mathbf{q})$ over the vector of outstanding shares $\mathbf{q}$. A trade is a movement $\mathbf{q} \to \mathbf{q} + \Delta\mathbf{q}$, and its cost is $C(\mathbf{q} + \Delta\mathbf{q}) - C(\mathbf{q})$. The gradient ∇C plays the role of prices; because C is a potential, total revenue depends only on the endpoints, not on the path.

The *logarithmic* MSR (LMSR) is the canonical choice. It yields the identity "price = probability," admits a closed-form bound on the maker's worst-case loss, and preserves path independence. In continuous-outcome settings (e.g., asset levels), naively splitting into many binaries fragments liquidity. We therefore discretize the outcome line into uniform *ticks* (bins) and apply LMSR to the whole line at once; traders buy *ranges* (unions of ticks) rather than single points. This preserves the LMSR guarantees while avoiding liquidity fragmentation.

To keep the mechanism viable on-chain at high resolution, we implement shares and prices with a lazy multiplicative segment tree, achieving $O(\log n)$ range updates and queries for n bins. Units are disciplined: externally we account in U6, internally all math uses WAD (18). A single conversion at the U6/WAD (18) boundary per trade, with buys rounded up and sells rounded down, eliminates trivial order-splitting arbitrage. Throughout, we take as baseline a venue with disjoint tick order books; range orders are executed via multi-leg routing unless a single centralized pricing potential couples ticks.

1.1 Scope and Conformance

This document is *normative* at the mechanism level (probability semantics, cost/pricing, tick semantics, rounding once per trade). Implementation details such as event names, function signatures, and ABI are *non-normative* and may evolve; a snapshot mapping for a reference implementation is provided in Appendix C.

2 Limitations of Disjoint Tick Order Books and the Case for an MSR

Order books over disjoint ticks are a natural baseline for continuous outcomes: let the outcome line be discretized into bins $\mathcal{B} = \{0, \dots, n-1\}$, and let each bin b have its own order book for the Arrow–Debreu state that pays 1 iff b realizes, with bin interval I_b. A user who wishes to trade an interval $[L, U)$ must then be routed into a multi-leg order across the set $\mathcal{B}(L, U) = \{b : I_b \subset [L, U)\}$.

2.1 Expressivity and execution risk

Multi-leg synthesis of ranges introduces (i) partial fills and non-atomic exposure across legs, (ii) gas costs and failure probability scaling with $|\mathcal{B}(L, U)|$, and (iii) execution slippage from price moves during routing. Providing a "one-click range order" therefore requires an internal rule that updates *multiple ticks simultaneously and atomically*, which already amounts to a centralized coupling of tick prices.

2.2 Global consistency: normalization and additivity

Let p_b denote the instantaneous price of bin b and $p(B) \equiv \sum_{b \in B} p_b$ for a tick-aligned range B. Absence of static replication arbitrage requires a probability interpretation:

$$\begin{aligned} &\text{(Normalization)} && \sum_b p_b = 1, \\ &\text{(Nonnegativity)} && p_b \geq 0, \\ &\text{(Additivity)} && p(B) = \sum_{b \in B} p_b. \end{aligned}$$

However, a venue with independently updated tick books lacks any *cross-book coupling* that preserves $\sum_b p_b = 1$ after fills/cancellations in one book. In practice, $\sum_b p_b$ will deviate from 1 over time unless other books are *simultaneously* adjusted. This lack of coupling leads to a quantitative failure of normalization:

Proposition 1 (Normalization breakdown under uncoupled ticks)**.** *In a system of disjoint tick order books without a coupling rule that preserves $\sum_b p_b$, there exist times at which $\left|\sum_b p_b - 1\right| \geq \varepsilon$ for some $\varepsilon > 0$. At such times, linear combinations of listed tick claims replicate payoffs at a cost that differs from their sum, enabling static replication arbitrage.*

Sketch. Induce net buying pressure in a single tick book so that p_b increases. In the absence of a mechanism that simultaneously decreases $\{p_j\}_{j \neq b}$ to offset the move, the sum $\sum_b p_b$ drifts away from 1. A portfolio long $\{b\}$ across all b replicates the full-range claim; any deviation of the sum from 1 yields a replication mispricing. □

2.3 Liquidity fragmentation

Liquidity is partitioned linearly across n tick books. If a minimal depth D is required per book for orderly execution, the venue demands $\Omega(nD)$ aggregate depth. With sparse order flow, spreads widen, unfilled orders accumulate, and multi-leg synthesis of ranges aggregates slippage across shallow books. By contrast, a single potential aggregates liquidity: every trade updates the global surface rather than only a local book.

2.4 Implication: centralized potential

To support one-click range orders *and* to maintain normalization/additivity globally, the venue needs a centralized pricing surface whose gradient defines tick prices and whose updates are atomic across the relevant ticks. This is precisely what a market-scoring rule provides. In CLMSR, the convex potential

$$C(\mathbf{q}) = \alpha \ln \Big(\sum_b e^{q_b/\alpha} \Big), \qquad p_b = \frac{e^{q_b/\alpha}}{\sum_j e^{q_j/\alpha}}$$

enforces p as a probability vector by construction. A range trade simply applies $w_b \leftarrow \varphi w_b$ for $b \in \mathcal{B}(L, U)$ with $\varphi = e^{\delta/\alpha}$, which updates the surface coherently and restores normalization instantly. Maker loss remains bounded by $\alpha \ln n$ (§9), and the implementation is viable on-chain via a lazy multiplicative segment tree (§7).

3 Units and Ticks

3.1 Outcome domain

We parameterize the market by

$$\text{OutcomeSpec} = (L, U, s, d),$$

with lower bound L, upper bound U, strictly positive tick size s, and metadata decimals d describing the raw oracle scale. The number of bins is *derived* as

$$n := \left\lceil \frac{U - L}{s} \right\rceil \geq 1.$$

Bins are indexed $b \in \{0, \dots, n-1\}$ with interior intervals $I_b = [L + bs, L + (b+1)s)$ for $b = 0, \dots, n-2$, and edge semantics handled by clamping (see below).

3.2 Units and ticks

CurrencyUnit is U6, the external 6-decimal settlement currency. **OutcomeUnit** is the raw oracle scale (e.g., price with 8 decimals, temperature in millidegrees). Let OutcomeRaw $= x$ denote the oracle reading in OutcomeUnit. The tick index is

$$b = \text{clamp}\left(\left\lfloor \frac{x - L}{s} \right\rfloor, \ 0, \ n-1 \right),$$

so that values below L map to the lowest bin and values at/above U map to the highest bin. Range trades specify $[L', U')$ in OutcomeUnit and implicitly affect the tick-aligned bin set B. We refer to δ as an *OutcomeUnit quantity* when it denotes a measurement size, and as a *CurrencyUnit amount* when it captures settlement notional; context will make the distinction explicit.

Unit separation. External debits, credits, and liquidity budgets are denominated in CurrencyUnit. All internal arithmetic—shares q_b, weights w_b, exponentials, and cached sums—uses WAD (18). A conversion factor of 10^{12} bridges the units. The liquidity parameter α carries CurrencyUnit economically but is stored in WAD (18), so q_b/α remains dimensionless.

Minimum order size. The core enforces a minimum trade size $\delta_{\min}$ in CurrencyUnit (default 10^{-2} U6). User interfaces must surface $\delta_{\min}$ together with the rounding wedge bound derived in Lemma 1.

Disclosure. Markets *MUST* publish (L, U, s, d) and the clamping rule so that off-chain services can reproduce the tick mapping.

4 Cost Function and Prices

4.1 Weights and partition sum

Define

$$w_b \equiv \exp\bigl(q_b/\alpha\bigr), \qquad \mathcal{Z} \equiv \sum_{b=0}^{n-1} w_b.$$

Initialize with $q_b = 0 \Rightarrow w_b = 1$, so $\mathcal{Z}_0 = n$ (with n derived from (L, U, s)).

4.2 CLMSR potential and instantaneous prices

The CLMSR potential (LMSR cost function) is

$$C(\mathbf{q}) = \alpha \ln\Bigl(\sum_{b=0}^{n-1} e^{q_b/\alpha}\Bigr) = \alpha \ln \mathcal{Z}.$$

The instantaneous price of bin b is the gradient component:

$$p_b = \frac{\partial C}{\partial q_b} = \frac{e^{q_b/\alpha}}{\sum_j e^{q_j/\alpha}} = \frac{w_b}{\mathcal{Z}}.$$

Hence *price = probability.* For a range B, the marginal price (per small share δ) is $\sum_{b\in B} p_b$.

5 Range Trading: Costs and Proceeds

5.1 Trade mechanics

Buying $\delta > 0$ shares of range B updates

$$\begin{aligned} q_b &\leftarrow q_b + \delta \quad (b \in B), \\ w_b &\leftarrow \varphi\, w_b \quad\;\; (b \in B), \qquad \boldsymbol{\varphi := e^{\delta/\alpha}}. \end{aligned}$$

Let $W_B = \sum_{b\in B} w_b$ and $W_{\bar{B}} = \mathcal{Z} - W_B$. Then

$$\mathcal{Z}_{\text{after}} = \mathcal{Z}_{\text{before}} + (\varphi - 1)\, W_B.$$

Reducing or closing a position of size $\sigma > 0$ over the same range applies the reciprocal factor $\varphi^{-1} = e^{-\sigma/\alpha}$, giving

$$\mathcal{Z}_{\text{after}} = \mathcal{Z}_{\text{before}} + (\varphi^{-1} - 1)\, W_B,$$

and the realized proceeds described below remain valid with this substitution. On chain we chunk large trades so that each exponential factor respects the guardrails `MIN_FACTOR` $\le \varphi^{\pm 1} \le$ `MAX_FACTOR` and never exceeds the pre-set `MAX_CHUNKS_PER_TX` limit.

From the potential, the buy cost and the sell/close proceeds are

$$\Delta C = \alpha \ln \frac{\mathcal{Z}_{\text{after}}}{\mathcal{Z}_{\text{before}}} \tag{1}$$

$$\Pi = \alpha \ln \frac{\mathcal{Z}_{\text{before}}}{\mathcal{Z}_{\text{after}}}. \tag{2}$$

Sequential buys and sells therefore commute: the proceeds from selling σ shares equal the cost that would have been paid to buy σ shares from the post-trade state. Positions that reach zero quantity burn their NFT representation, matching the core contract's ‘decreasePosition‘ path.

Proposition 2 (Binary reduction for ranges)**.** *For any range B, the trade behaves exactly like a binary LMSR over the partition $\{B, \bar{B}\}$ with initial masses $(W_B, \mathcal{Z}_{\text{before}} - W_B)$:*

$$\Delta C = \alpha \ln\Bigl(1 + (\varphi - 1)\frac{W_B}{\mathcal{Z}_{\text{before}}}\Bigr).$$

Proof. Substitute $\mathcal{Z}_{\text{after}} = \varphi W_B + (\mathcal{Z}_{\text{before}} - W_B) = \mathcal{Z}_{\text{before}} + (\varphi - 1) W_B$ into the definition of ΔC. □

5.2 Consistency checks

When B spans all bins, a trade of size δ costs exactly δ. In the small-trade limit $\delta \to 0$, the average price $\Delta C/\delta$ approaches the marginal $\sum_{b\in B} p_b$. External accounting routes through the shared rounding helpers exactly once per trade, applying the prescribed round-up/round-down policy, and the binary reduction follows from Proposition 2.

6 Tick Mapping and Settlement

Settlement consumes the raw oracle value OutcomeRaw. We map it to a bin via

$$\text{toTick}(x) = \min\Bigl\{n - 1,\ \max\Bigl\{0,\ \Bigl\lfloor \frac{x - L}{s} \Bigr\rfloor\Bigr\}\Bigr\}.$$

This *edge-clamped* mapping ensures that realizations outside $[L, U)$ are attributed to the corresponding edge bins.

The core exposes a settlement entrypoint that consumes OutcomeRaw and maps it to the realized tick via the helper above. On settlement, the system *MUST* emit events that (i) record the *market-level* realization (tick index and, optionally, the raw value) and (ii) record *position-level* payouts. Event names and ABI are implementation-specific and thus non-normative (Appendix C). A position (L, U, δ) pays δ CurrencyUnit iff the realized tick lies in $B(L, U)$. Settlement parameters ($s > 0$, $n > 1$, tick alignment) are validated at market creation. Front-ends should display the range semantics and link to the canonical mapping so that off-chain monitoring reproduces the realization.

7 Lazy Multiplicative Segment Tree

7.1 Data layout

We maintain an array-based segment tree over bins. Each node keeps the exact subtree sum in WAD (18), a lazy-multiplication tag (initially 1), and packed child pointers so that untouched subtrees can be materialised later. Leaves default to $w_b = 1$, so idle ranges need no storage yet still contribute a unit weight to $\mathcal{Z}$.

7.2 Core operations

Range multiplication `rangeMul`(b_ℓ, b_u, φ) multiplies both `sum` and `pendingFactor` by φ along covered nodes, realizing $w_b \leftarrow \varphi w_b$ for all $b \in [b_\ell, b_u]$ in $O(\log n)$ and deferring leaf updates via the multiplicative lazy tag. The helper auto-allocates child nodes the first time a partial overlap occurs, ensuring sparsity for dormant bins.

Range sum `rangeSum`(b_ℓ, b_u) pushes tags as needed and returns $\sum_{b=b_\ell}^{b_u} w_b$.

Root cache $\mathcal{Z}$. $\mathcal{Z}$ is *consistently exact under WAD*: updates follow fixed-point rounding rules. The update order is (i) measure W_B before applying the trade factor φ, (ii) compute $\Delta C = \alpha \ln(1 + (\varphi - 1)W_B/\mathcal{Z})$, (iii) apply the lazy range multiplication, (iv) update $\mathcal{Z} \leftarrow \mathcal{Z} + (\varphi - 1)W_B$, and (v) perform a single CurrencyUnit conversion for external transfer. The helper keeps `root.sum` and $\mathcal{Z}$ in sync, giving $O(1)$ access to the partition sum.

7.3 Invariants and complexity

Each node's stored sum equals the exact sum of its leaves' w_b times all pending factors on the path, and lazy tags are cleared whenever they are pushed into children. The root cache maintains $\mathcal{Z} =$ `root.sum` at all times so trade cost/proceeds can be derived without additional queries. If W_B is measured before applying φ on B, the update $\mathcal{Z} \leftarrow \mathcal{Z} + (\varphi - 1)W_B$ keeps $\mathcal{Z}$ exact and matches the cost expression. Each range update or sum touches $O(\log n)$ nodes; concretely, at most $4\lceil \log_2 n \rceil$ in our layout (a typical conservative bound for array-backed trees), and batching adjacent ranges further reduces constants.

8 Numerical Safety and Rounding

This section lists the guardrails that make the fixed-point implementation robust in practice.

8.1 Exponentials and chunking

To compute $\varphi = e^{\delta/\alpha}$ safely, inputs are chunked below a fixed bound `MAX_EXP_INPUT_WAD`; the number of chunks per transaction is capped for gas-DoS resilience. Each chunk respects the allowable exponent window, and the loop fails fast if the requested split would exceed `MAX_CHUNKS_PER_TX`.

8.2 Factor bounds and auto-flush

We enforce `MIN_FACTOR` $\le \varphi \le$ `MAX_FACTOR` and automatically push lazy tags when a node's `pendingFactor` exceeds `FLUSH_THRESHOLD`, preventing loss of significance on heavily-traded ranges.

8.3 Rounding discipline

Rounding Policy. All internal computations use WAD (18). External transfers *convert to* U6*once per trade*: buys use $\lceil\cdot\rceil$ (round-up) and sells use $\lfloor\cdot\rfloor$ (round-down). This policy prevents split-order rounding gains and bounds the discrepancy between observed fills and internal probabilities.

Theorem 1 (No split-order rounding arbitrage)**.** *For any partition $\delta = \sum_i \delta_i$, $\sum_i \lceil \Delta C_i \rceil \ge \lceil \Delta C \rceil$ (buy) and $\sum_i \lfloor \Pi_i \rfloor \le \lfloor \Pi \rfloor$ (sell).*

Sketch. Subadditivity and superadditivity of the ceiling and floor operators give the inequalities immediately once costs and proceeds are accumulated in WAD (18) before the single final conversion. $\square$

Lemma 1 (Observed price wedge)**.** *Let δ denote the executed trade size in CurrencyUnit. Then the difference between the observed average fill price (post-rounding) and the internal LMSR probability is bounded by $|p_{\text{obs}} - p_B| \le 10^{-6}/\delta$. At $\delta = \delta_{\min}$ the bound is $10^{-6}/\delta_{\min}$, which is surfaced to users.*

Single-rounding invariant. Every execution path funnels through shared helpers (implementation-specific) that enforce a single conversion per trade under the prescribed rounding direction, ensuring that no intermediate loop performs premature conversions. The invariant is monitored in tests and by gas-guard checks that the helpers are called at most once per trade.

8.4 Fixed-point error budget

A trade visits $O(\log n)$ WAD operations; each operation may incur at most one ULP of error (10^{-18}). Consequently the relative error across $\mathcal{Z}$ and W_B

is bounded by $c \log n \cdot 10^{-18}$ for an implementation-dependent constant c, and the cached sums stay in lock-step with the tree state.

9 Risk and Loss Bound

Theorem 2 (Loss bound)**.** *The market maker's worst-case loss does not exceed* $\alpha \ln n$*.*

Sketch. Starting from $\mathbf{q} = \mathbf{0}$ with $C(\mathbf{0}) = \alpha \ln n$, cumulative maker revenue equals $C(\mathbf{q}_T) - C(\mathbf{0})$. If tick i^* realizes, the payout is $q_{i^*}(T)$. Convex duality for $\log\sum\exp$ yields

$$q_{i^*}(T) - \big(C(\mathbf{q}_T) - C(\mathbf{0})\big) \leq \alpha \ln n,$$

so the realized loss is at most $\alpha \ln n$. □

Guideline. For a target loss budget B, choose $\alpha = B/\ln n$. Finer ticks (larger n) improve fidelity but increase the ceiling $\alpha \ln n$ and the tree depth. Here n is the derived bin count $\lceil (U - L)/s \rceil$.

10 Economic Considerations

Protocol-level fees should be surfaced explicitly rather than implicitly baked into ΔC: folding a fee into the potential breaks the *price = probability* identity and complicates hedging. We therefore add fees after the rounding helpers, preserving LMSR semantics and making the markup transparent.

Settlement boundaries follow the half-open convention $[L + bs,\, L + (b + 1)s)$ (with optional closure at the top boundary). Settlement events that record the realized tick (and optionally the raw value) enable auditors to replay payouts off-chain using the published `OutcomeSpec`; event names and ABI are implementation-specific (Appendix C).

11 Mechanism-Level Security

Market creation and settlement are owner-gated, and only the core may mutate positions. The core is pausable and ordered for checks–effects–interactions, with reentrancy guards on external transfers. Inputs are validated for tick alignment, range bounds, factor bounds, and chunk limits; violations revert. External accounting funnels through shared rounding helpers once per trade in accordance with the prescribed policy (§8).

12 Contract Surface (Core Only)

The mechanism maps to the following contract surface: **CLMSRMarketCore** manages the segment tree, computes costs/proceeds, handles external token accounting, and enforces the market lifecycle. It uses UUPS upgradeability, pausability, and reentrancy protection.

CLMSRPosition is an ERC-721 representing $(\text{marketId}, L, U, \text{shares})$ with `onlyCore` controls.

Lifecycle. A market is created with an `OutcomeSpec` tuple (L, U, s, d) alongside trading window and liquidity parameters. During trading, users may open, increase, decrease, and close positions subject to `MIN_ORDER_U6` $= 10^{-2}$ U6. After expiry, the owner posts the realized oracle reading via a settlement entrypoint, which resolves to a tick index through the helper in §6 and emits market- and position-level settlement events (names/ABI non-normative; Appendix C). A position (L, U, δ) pays δ U6iff the realized tick is in $B(L, U)$.

13 Parameter Choices and Practical Notes

Smaller tick spacing s increases fidelity but also $\ln n$ and tree depth, affecting both the maker's loss ceiling and gas constants. A larger α flattens the curve, deepening the book and reducing slippage while raising the loss ceiling linearly. Quick consistency checks: (i) ensure a full-range buy of δ costs exactly δ; (ii) verify that sequential buys on the same range equal a single buy of the sum; (iii) for small δ, check $\Delta C/\delta \approx \sum_{b \in B} p_b$; (iv) after settlement, confirm total payouts equal the number of winning shares.

Choosing (L, U, s)**.** Operators may set (L, U) using volatility heuristics (e.g., multiples of ATR) and then choose s to balance resolution and gas costs. Because $n = \lceil (U - L)/s \rceil$ is derived, tighter spacing increases n (and thus $\alpha \ln n$ and tree depth) while improving fidelity near the boundaries. Edge-clamped settlement ensures tail realizations accrue to the edge bins by construction.

14 Worked Example

Let $n = 8$, $w_b = 1$, $B = \{2, 3, 4\}$, $\alpha = 50\,\text{U6}$ (internally $50 \cdot 10^{18}$), and buy $\delta = 10$ shares (internally $10 \cdot 10^{18}$). Then $\delta/\alpha = 0.2$, so $\varphi = e^{0.2} \approx 1.221402758$. With $W_B = 3$ and $\mathcal{Z} = 8$,

$$\begin{aligned}\Delta C &= \alpha \ln\Big(1 + 0.221402758 \cdot \tfrac{3}{8}\Big) \\ &\approx 3.98\,\text{U6}.\end{aligned}$$

The external debit routes through the shared rounding helper. If `settlementBin` $\in B$, the position pays $\delta = 10\,\text{U6}$.

15 Related Work

The CLMSR presentation builds on the logarithmic market scoring rule introduced by Hanson [1] and the bounded-loss analysis of Chen and Pennock [2]. Abernethy et al. [3] and Peters et al. [4] describe convex-potential market makers and sparse data structures closely related to our lazy tree. On-chain adaptations drawing from continuous outcomes include Othman et al. [5] and recent DeFi deployments [6] that motivate our unit discipline and rounding constraints.

16 Conclusion

CLMSR implements continuous-outcome prediction on-chain with a single convex potential, range trading, and a clean loss ceiling. Ticking the domain preserves tractability while allowing high resolution; a lazy multiplicative segment tree gives logarithmic-time updates and queries. Strict unit discipline (U6external, WAD (18) internal), a one-time rounding policy per trade, and simple numerical guards make the mechanism production-friendly without compromising the LMSR properties. This specification focuses on the mechanism itself; operations, token economics, and off-chain infrastructure are left to separate documents.

References

References

[1] R. Hanson. Combinatorial information market design. *Information Systems Frontiers*, 5(1):107–119, 2003.

[2] Y. Chen and D. Pennock. A utility framework for bounded-loss market makers. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 49–56, 2007.

[3] J. Abernethy, V. Syrgkanis, and M. Tewari. Efficient market making via convex potentials. In *Proceedings of the Fourteenth ACM Conference on Electronic Commerce*, pages 683–700, 2013.

[4] M. Peters, K. Leyton-Brown, and N. Sandholm. Sparse updates for conditional market scoring rules. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 602–611, 2012.

[5] A. Othman, D. Pennock, D. M. Reeves, and T. Sandholm. Automated market making for complex securities over continuous outcomes. In *Proceedings of the Fourteenth ACM Conference on Electronic Commerce*, pages 395–412, 2013.

[6] R. Xu, T. Le, and L. Jiang. Automated market makers for decentralized finance. *arXiv preprint arXiv:2102.11345*, 2021.

Disclaimer

A Symbol Table

Symbol	Meaning
α	Liquidity parameter (U6 unit; internal WAD (18))
n	Number of bins (derived $= \lceil (U-L)/s \rceil$)
s	Tick size in OutcomeUnit
L	Minimum outcome in OutcomeUnit
q_b	Shares at bin b (1 share pays 1 U6 if b realizes)
w_b	Weight $e^{q_b/\alpha}$ (WAD (18))
$\mathcal{Z}$	Partition sum $\sum_b w_b$ (WAD (18))
$[L, U)$	Purchased tick range (tick-aligned)
δ	Shares bought/sold over B (internal WAD (18); external U6)
$\delta_{\min}$	Minimum trade size enforced by the core (U6)
φ	$e^{\delta/\alpha}$ (multiplicative factor on B)
ΔC	Buy cost (Eq. (1))
Π	Sell proceeds (Eq. (2))
p_b	Instantaneous price/probability $w_b/\mathcal{Z}$
W_B	Range weight $\sum_{b\in B} w_b$
OutcomeRaw	Oracle observation in OutcomeUnit
`OutcomeSpec`	(L, U, s, d) settlement metadata

B Implementation Constants

Constant	Purpose
`MAX_EXP_INPUT_WAD`	Bound for exponential input (overflow/precision)
`MAX_CHUNKS_PER_TX`	Bound on chunked exponential segments
`MIN_FACTOR`, `MAX_FACTOR`	Bounds on multiplicative factors
`FLUSH_THRESHOLD`	Threshold for auto-flush of lazy tags
`MAX_TICK_COUNT`	Upper bound on derived bins per market

C Reference Implementation Snapshot (Non-Normative)

This appendix documents an implementation snapshot to aid integrators. It is informative only; names and ABI may change without affecting the normative mechanism-level specification.

OutcomeSpec mapping

Spec symbol	Typical implementation field
L (minimum outcome)	*e.g.* `minTickLike`
U (maximum outcome)	*e.g.* `maxTickLike`
s (tick size)	*e.g.* `tickSpacingLike`
d (raw scale meta)	*e.g.* `decimals`
n (derived)	stored as $\lceil (U-L)/s \rceil$ for initialization

Settlement events (roles)

- Market-level settlement: publishes the realized tick (and optionally the raw value).
- Position-level settlement: publishes per-position payout/closure.

Rounding policy

Deployments apply the fixed policy described in §8: buys round up, sells and settlements round down, and exactly one conversion occurs per trade. Under this configuration, Theorem 1 and Lemma 1 hold.

D Selected Interfaces (Core Only)

- **Market:** `createMarket`, `settleMarket`
- **Positions:** `openPosition`, `increasePosition`, `decreasePosition`, `closePosition`, `claimPayout`
- **Views:** `getMarket`, `getRangeSum`, `isPaused`. Single-bin probabilities/prices are derived via `getRangeSum` to obtain the bin weight and $\mathcal{Z}$, then compute $p_b = w_b/\mathcal{Z}$.
- **Tree:** `applyRangeFactor`